Activity 1

The store under consideration is assumed to be a small grocery store with a staff of 20 people including the manager. Because of the limited scope of the assignment, the only interaction between the customer and the manager is assumed to be stemmed from the customer making a complaint to the manager. The customer also stays anonymous as its only role is to make a complaint without sharing his name aside from the buying of items. The business has yet to run loyalty programs which necessitate the storing of information of customers. It is also assumed that every time the customer makes a complaint, the manager can always resolve his complaint as per the recommendation to consider only the success case by Tabinda). As the manager is assumed to own the business, she does not get paid the same way that cashiers do. In this store, except for the manager, everyone's title is cashier as they all need to serve at a till. There is also only one manager in the store. Because InventorySystem and TimeTrackingSystem share common behaviour, they will inherit from the abstract class BasicSystem assumed to interact with the GUI and perform other expected operations of such a system, all of which are not included as they are outside of the scope of the assignment. All the classes could contain more attributes and operations that would be necessary in real-life applications, but they only contain minimal attributes and operations congruous with what is expected of dummy code.

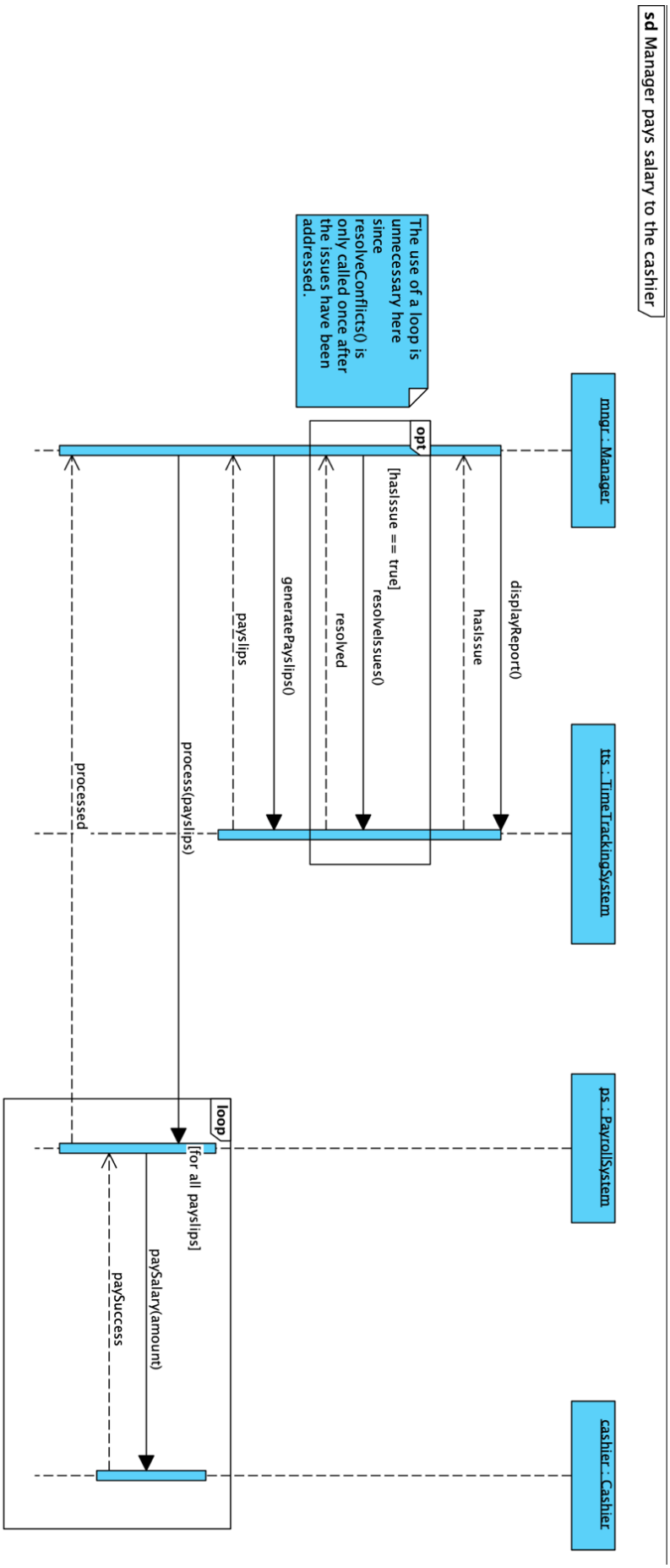
Activity 1-1 – Manager pays salary to the cashier

Assumptions and explanations (these may not be directly reflected in the corresponding sequence diagram): The time tracking system takes in a time schedule either manually prepared by the manager or automatically created by the time scheduling system itself and then it keeps track of the clock-ins and outs of the employees (cashiers). The aforementioned procedure is not directly related to the use case "Manager pays salary to the cashier". Therefore, the use case begins in the sequence diagram with all these actions assumed to have taken place prior.

The business has the following procedure for paying its employees:

1. The manager requests the time tracking system to display its internal report.
2. The tracking system's report is displayed on the manager's UI.
3. If any time conflict between the time schedule and records of employee clock-ins and outs from the time tracking system happens to exist in the report, the manager resolves them (the details of how the conflicts are resolved are too low-level to be described in the sequence diagram, although they will be resolved through confirming with cashiers, etc). Once the manager resolves all the discrepancies, she then updates the time tracking system with modified information in the form of invoking resolveConflicts(), after which the system displays an updated report again.
4. If no time conflict exists, the manager approves of the report by submitting it to the payroll system via the time tracking system, which generates payslips with which the payroll system can pay cashiers correctly.
5. The payroll system takes the given payroll and pays each cashier employee on its system with the amount derived from the information stored in the time tracking system.

S3774430 Myeonghoon Sun



S3774430 Myeonghoon Sun

Activity 1-2 – *Manager looks after the inventory of the store*

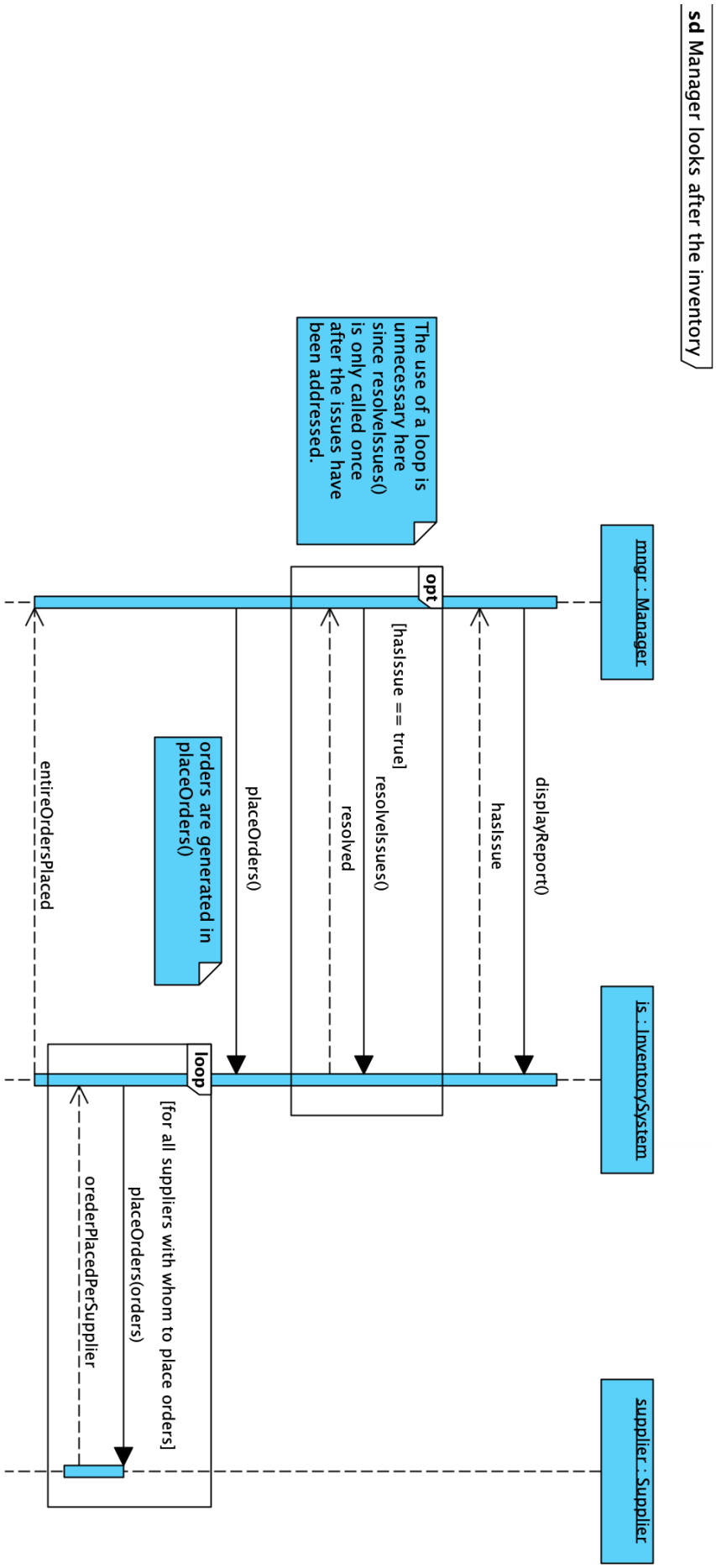
Assumptions and explanations (these may not be directly reflected in the corresponding sequence diagram): First, because of the ambiguity of what is meant by "look after the inventory" without any context provided, it is important to clearly point out what the action of looking after the inventory includes. The phrase "look after" has a strong implication that the associated action is carried out on a regular basis. Given this common-sense definition, it is assumed that looking after the inventory refers to day-to-day operations rather than large-scale stock counts before a major stocktake.

Given the assumed definition of looking after the inventory, it is also as important to delineate what constitutes the manger's responsibility and what does not when it comes to looking after the inventory. Here in the store, the manager's sole responsibility of looking after the inventory is to resolve issues the inventory system flags (e.g. the given product's stock is too low or too high [resolving issues can come in a great many forms. Thus, they are effectively abstracted into a function called resolveIssues() in the sequence diagram]) since the inventory system keeps track of the number of items per product and suggests orders that need to be placed; It is the assumed business logic the manager needs to resolve inventory-system-generated issues before being able to proceed to place orders. Thanks to this automated system, the manager only needs to approve of the suggested orders so that they can be placed with corresponding suppliers. Moreover, it is assumed that activities including but not limited to product storage management, product rotation, receiving of delivery are the responsibilities of regular staff members, since they are physical tasks as often as not performed by regular staff members in real life. The manager may give instructions on how each of the tasks should be performed, but these activities should be covered in use cases "The employee looks after the inventory" and "The manager manages staff", respectively.

The business has the following procedure for the manager to look after the inventory of the store:

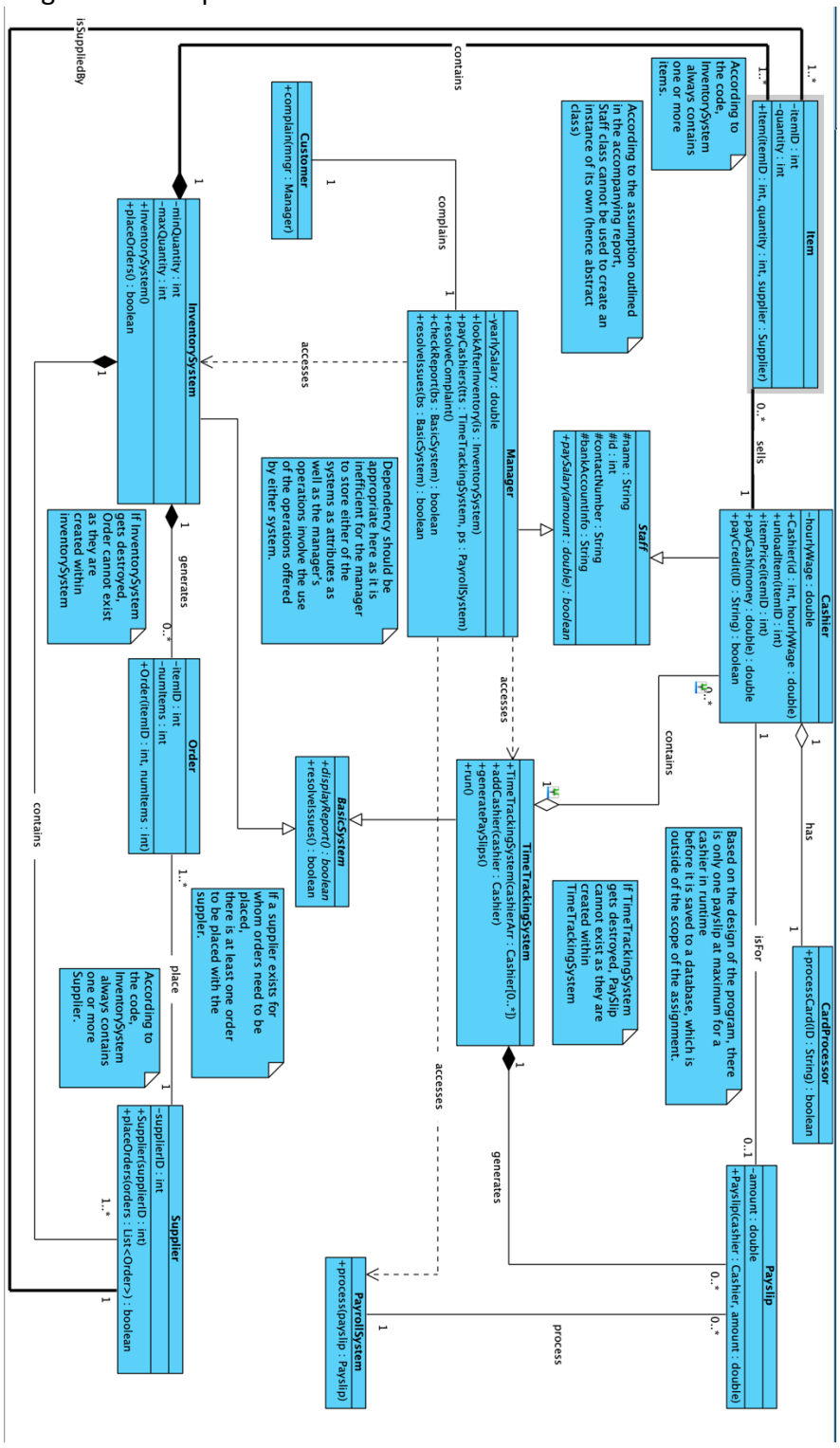
1. The manager requests a report to be displayed from the inventory system.
2. The inventory system generates a report with suggested orders and the manger's UI displays it.
3. If there are flagged issue(s) in the report, the manager needs to resolve them, after which the inventory system generates another report with suggested orders to refill the inventory.
4. The manager approves of the report by placing the suggested orders with corresponding suppliers (within placeOrders(), order objects are created; this part is assumed to happen within the function, therefore it is not depicted in the sequence diagram since it should not be concerned with such low-level details).

S3774430 Myeonghoon Sun

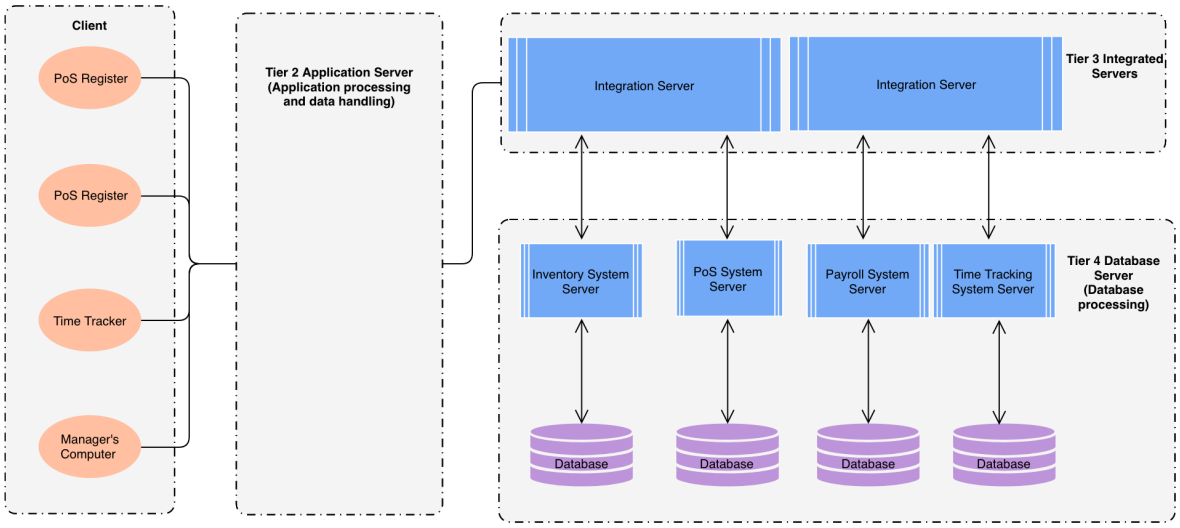


Activity 2

The specification reads “Create a main class for instantiating the classes and run all the methods described **in your solution of activity 1**. My solution in code addresses three of the specified requirements. Therefore, no code corresponding to the provided sequence diagram in the specification is written in the submitted Java code.



Activity 3



Given the requirements for a given store to have access to inventory, payroll and PoS systems, and its assumed enterprising goal to scale up by opening more stores in the future, the 4-tier client-server architecture is recommended for the following reasons.

Because the store in question does not intend to outreach business in bricks and mortar by establishing its e-commerce presence on the internet, the architecture model involving the Model-View-Controller is deemed rather excessive for relatively smaller applications like the one the store needs.

According to Sommerville (Sommerville, 2015), because of the store's grand goal to scale out, the two-tier-client-server approach is going to cause problems with scalability and performance or problems of system management, unlike the ease of adding servers with a multi-tier client-server architecture that is well-suited for large-scale, or to-be-large-scale, applications with a great many clients. He goes on to mention that a multi-tier client-server architecture is effective for an application where the data are volatile. For example, in our store scenario, data stored in the database connected to the PoS system are going to go through countless transformations (volatile). Not only that, he writes that a multi-tier system may also be used when an application needs to access and use data from different databases, which can be integrated into an additional server as seen in the diagram above between the tier 3 and 4.

The multi-tier architecture approach comes with the additional benefit of rapidly responding to client requests as processing is distributed across multiple tiers, which is an important non-functional requirement to meet as slow-processing time on a PoS system could detract from customer satisfaction. As seen in the diagram above, multiple types of clients interact with the tier 2 application server where the majority application processing happens before data are requested. When the requested data are sent back, the existence of the application server can unload the amount of work the other tiers would have otherwise carried out. Therefore the 4-tier client-server as depicted in the diagram is recommended as the architecture of the software solution for the store.

Reference

- Sommerville, I., 2015. *Software Engineering*. 10th ed. Hallbergmoos: Pearson Education Deutschland GmbH, p.505, 506.